Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Aleš Turk

Grajenje spletnih strani v spletnem brskalniku

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: doc. dr. Aleš Smrdel

Ljubljana, 2017

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

V današnjem času je skoraj obvezno, da ima vsak tudi spletno stran, ki pa jo je potrebno narediti in gostiti na strežniku. Izdelava spletnih strani pa ni preprosta in zahteva čas ter poznavanje spletnih tehnologij. Orodje, ki bi omogočalo vizualno gradnjo spletnih strani, pa bi lahko znatno olajšalo samo izdelavo. Vaša naloga je, da v okviru diplomskega dela razvijete spletno aplikacijo, ki bo omogočala vizualno gradnjo spletnih strani in od uporabnika ne bo zahtevala poznavanja tehnologij, obenem pa bo intuitivna za uporabo. Aplikacija naj omogoča določanje vsebine, slogovno oblikovanje ter spreminjanje strukture elementov. Pri izdelavi se osredotočite na dobro uporabniško izkušnjo in odzivnost aplikacije na uporabniške akcije. V okviru diplome najprej določite funkcionalnosti, ki jih mora tovrstna aplikacija obsegati. Nato preučite tehnologije na strani strežnika in na strani odjemalca, ki so na voljo, in izberite najprimernejše za razvoj zahtevane aplikacije.

Zahvaljujem se svoji celotni družini za vso spodbudo, motivacijo ter finančno podporo v času mojega študija. Zahvalil bi se tudi mentorju doc. dr. Alešu Smrdelu za mentorstvo in koristne nasvete pri izdelavi diplomskega dela.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
HTML	hyper text markup language	jezik za označevanje nadbesedila
CSS	cascading style sheets	kaskadne stilske predloge
SASS/ SCSS	syntactically awesome stylesheets	izboljšane kaskadne stilske predloge
SQL	structured query language	strukturiran poizvedbeni jezik
JSON	JavaScript object notation	objektni zapis JavaScript
XML	extensible markup language	razširljivi označevalni jezik
HTTP	hypertext transfer protocol	protokol za prenos informacij
URL	uniform resource locator	enolični krajevnik vira
SPA	single page application	enostranska aplikacija
SaaS	software as a service	programska oprema kot storitev

Povzetek

Naslov: Grajenje spletnih strani v spletnem brskalniku

Namen diplomske naloge je uporabnikom, ki nimajo predhodnega znanja spletnih tehnologij oziroma znanja programiranja, omogočiti izdelavo lastne spletne strani. Pri razvoju spletne aplikacije je bilo potrebno najprej izbrati najprimernejše tehnologije za izdelavo same aplikacije. Nato je bilo potrebno določiti potrebne funkcionalnosti, čemur je sledila implementacija aplikacije. Z razvito aplikacijo lahko uporabnik preko uporabniškega vmesnika vnese vsebino, stilsko oblikuje izgled in spremeni strukturo komponent (teme), ki v realnem času spremeni izgled izdelane spletne strani, kar posledično omogoča hitrejši in enostavnejši razvoj, saj uporabniku ni potrebno osveževati strani ob vsaki najmanjši spremembi. Aplikacija omogoča hitro gradnjo lastnih spletnih strani kar preko brskalnika, s tem pa uporabniku prihrani čas ter finančna sredstva, ki bi jih moral vložiti v razvoj lastne spletne strani.
Aplikacija, ki je bila razvita v okviru diplome, je namenjena predvsem manjšim podjetnikom, ki želijo svojo storitev tržiti preko spleta, blogerjem, predstavitvam izdelkov in predstavitvam dogodkov.
Razviti prototip predstavlja izhodišče za razvoj večjega sistema SaaS (software as a service).

Ključne besede: grajenje spletne strani, spletni brskalnik, MEAN, Angular.

Abstract

Title: Building websites in a web browser

The aim of the thesis is that the users, who have no prior knowledge about web technologies or have no programming skills, can create their own website. When developing the web application we had to select the most appropriate technologies for development first. Next, we had to determine the required functionality, which was followed by the development of the application.. The application offers an user interface, where the user can enter the content, design the appearance and change the structure of the components (themes). The changes are applied in real time, which enables a faster and easier workflow while creating the website, with no page reload needed for each change. Users can build their own website through the browser, which saves them the time and funds required, which would otherwise have to be invested in the development. The application, which was developed in the scope of the thesis, is primarily meant for small business owners, who would like to market their service on the internet, bloggers, product presentations and event presentations.
The developed application serves as a prototype of a larger software as a service system.

Keywords: building website, web browser, MEAN, Angular.

Poglavje 1

Uvod

Uporaba spletnih tehnologij oziroma spletnih strani je v današnjem času vse bolj pogosta, nekateri bodo celo rekli, da je nujno potrebna. Obstaja nekakšno nepisano pravilo, da če nimaš svoje spletne strani, potem ne obstajaš. To še posebej velja za podjetja in ljudi, ki se ukvarjajo z določeno obrtjo. Veliko ljudi uporablja Internet za iskanje raznoraznih informacij in možnosti o storitvah različnih ponudnikov. Spletne strani pa je seveda potrebno tudi zgraditi in objaviti, kar pa zahteva vsaj osnovno poznavanje spletnih tehnologij. V ta namen je bilo v okviru diplomskega dela razvito orodje, ki uporabnikom omogoča hitro in enostavno grajenje spletnih strani brez predhodnega znanja programskih jezikov ali poznavanja spletnih tehnologij.

1.1 Cilji diplomske naloge

Ciljev diplomske naloge je več. Prvi in glavni cilj je uporabnikom brez znanja spletnih tehnologij omogočiti izdelavo svoje lastne spletne strani preko uporabniškega vmesnika v spletnem brskalniku. Uporabnik bo v realnem času videl spremembe, ki jih je vnesel, in na tak način dobil boljšo predstavo o končni spletni strani. S takim sistemom bodo uporabniki, kot so na primer avtomehaniki, lepotni saloni in organizatorji dogodkov lahko izdelali predstavitveno spletno stran, ne da bi v razvoj vlagali veliko denarja in časa.

Tudi ostali uporabniki, kot so na primer blogerji in pisci člankov, bodo lahko naredili spletno stran prilagojeno svojim potrebam. Uporabniki bodo imeli veliko možnosti za spremembo vsebine in izgleda strani. Drug pomemben cilj pa je tudi pridobiti čim več znanja in izkušenj v spletnih tehnologijah.

1.2 Struktura diplome

V prvem poglavju je predstavljena tema diplomske naloge. Naslednje poglavje opisuje zahteve aplikacije. Sledi kratka predstavitev uporabljenih tehnologij. Ker je bila pri razvoju uporabljena najnovejša različica ogrodja Angular, ki je še dokaj nova in ne preveč dobro opisana, je naslednje poglavje namenjeno podrobnejši predstavitvi te različice ogrodja Angular. Temu poglavju sledi še predstavitev razvoja, na koncu pa so predstavljeni še zaključki in ideje za nadaljnje delo.

Poglavje 2

Funkcionalne zahteve in izbor tehnologij

Razvoj spletne aplikacije je potekal v več korakih. Najprej smo določili funkcionalne zahteve, ki so bile nujno potrebne za pravilno in učinkovito delovanje aplikacije. Ko so bile zahteve določene, je bilo potrebno izbrati tehnologije in razvojno okolje za razvoj aplikacije.

2.1 Funkcionalne zahteve

Pod funkcionalne zahteve spadajo funkcionalnosti, ki so potrebne, da bo aplikacija delovala kot storitev za druge uporabnike. Spletna aplikacija je bila razdeljena na tri večja področja: uporabnike, plačilni sistem in uporabniški vmesnik, ki bo uporabnikom omogočal izdelavo lastne spletne strani.

Zahteve, ki so povezane z uporabniki, so:

- **registracija** - kot že ime funkcionalnosti pove, moramo v storitev vpeljati registracijo uporabnikov.
- **prijava** - prijava poteka z uporabniškim imenom (elektronski naslov) in geslom, ki si ga je uporabnik določil ob registraciji. Ko se uporabnik prijavi, dobi dostop do plačilnega sistema in izdelovanja aplikacij.

- **odjava** - po uspešno izvedeni odjavi, se uporabnika preusmeri na začetno stran.

Pod plačilni sistem spada izbira različnih paketov, ki uporabniku nudijo različno količino spletnih strani, ki jih lahko ustvari. Ko uporabnik izbere paket, se mu prikaže plačilni obrazec, kamor vnese podatke o kartici.

Najbolj obsežen del aplikacije je uporabniški vmesnik in kreiranje aplikacij. Pod to rubriko spadajo:

- **kreiranje rešitve** - uporabnik lahko kreira svojo lastno spletno stran (določi ime).
- **spreminjanje rešitve** - ko je uporabnik že naredil rešitev, lahko to rešitev predela po svoji želji. K predelavi spadajo spreminjanje vsebine, spreminjanje izgleda in dodajanje dodatnih strani.
- **brisanje rešitve** - če uporabnik s svojo rešitvijo ni zadovoljen ali pa je več ne potrebuje, jo lahko enostavno izbriše.

2.2 Izbira tehnologij

Ko so bile funkcionalne zahteve določene, je bilo potrebno poiskati tehnologije, ki bi bile najbolj primerne za izdelavo spletne aplikacije. Potrebna je baza podatkov za uporabnike in njihove rešitve, jezik na strežniku (ang. back-end language) za komunikacijo med uporabniškim vmesnikom in bazo ter jezik uporabniškega vmesnika (ang. front-end language). Obstaja veliko različnih načinov in ogrodij, ki omogočajo lažje razvijanje. Mi smo se odločili za tehnologije MEAN, saj vključujejo vse tehnologije, ki jih potrebujemo. MongoDB poskrbi za shranjevanje podatkov v podatkovno bazo, Express.js in Node.js poskrbita za komunikacijo med uporabniškim vmesnikom in podatkovno bazo, Angular pa poskrbi za interakcijo uporabnika z uporabniškim

vmesnikom. Edino vprašanje, ki se je tu porodilo, je, ali izberemo zelo popularno različico ogrodja Angular.js ali novejšo različico Angular2. Odločili smo se za novejšo različico, saj je hitrejša in omogoča boljši nadaljni razvoj.

2.3 Izbira razvojnega okolja

Za integrirano razvojno okolje (ang. integrated development environment, kratica IDE) smo izbrali WebStorm podjetja JetBrains. Na izbiro smo imeli veliko primernih orodij, kot so Visual Studio, Visual Studio Code in Atom, vendar smo se odločili za WebStorm. Prepričala sta nas predvsem vgrajen terminal (konzola) ter dobra podpora jeziku TypeScript, ki smo ga uporabili za pisanje Angular kode.

Poglavje 3

Uporabljene tehnologije

V tem poglavju so opisane tehnologije, ki so bile uporabljene pri izpolnjevanju zadanih ciljev.

3.1 HTML

Hyper Text Markup Language (jezik za označevanje nadbesedila) je standardni jezik za izdelavo spletnih strani [1]. Uporablja se za opisovanje strukture spletne strani s pomočjo oznak (html, body, h1, div, p, ...). Brskalniki teh oznak ne prikažejo, izpišejo le vsebino, ki se nahaja med oznakami. Posameznim oznakam je dodana tudi stilska vsebina, ki se uporablja, ko brskalniki izpišejo vsebino, ki se nahaja med oznakami. Ta stilska vsebina lahko vpliva na izgled ali strukturo oznake. Oznake imajo ponavadi privzeto stilsko vsebino že vnaprej definirano. Ta stilska vsebina se lahko na različnih brskalnikih tudi razlikuje. Tako ima na primer oznaka $<h1>$ že privzeto določeno velikost in odebeljenost pisave ter odmike (ang. margin). Oznakam se lahko dodajo tudi atributi, ki slepim in slabovidnim preko bralnikov zaslona (ang. screen reader) pomagajo razumeti vsebino spletne strani.

Dokument HTML je sestavljen iz treh delov, prikazanih tudi v kodi 3.1:

- **DOCTYPE** - predstavlja tip dokumenta, ki brskalniku pomaga pravilno prikazati spletno stran. Naveden mora biti na vrhu pred oznako

$<html>$, ki predstavlja korenski element dokumenta HTML.

- **head** - vsebujejo meta podatke, ki običajno niso prikazani, so pa pomembni za interpretacijo dokumenta. Vsebuje tudi informacije za prikaz strani ($<style>$, $<script>$), oziroma pot ($<link>$) do zunanjih datotek (ponavadi .css). Te datoteke so lahko shranjene na disku ali pa na zunanjem viru podatkov, dosegljivem preko URL.

- **body** - vsebuje strukturo strani. Ta del je najbolj pomemben za prikaz strani, saj so tu navedeni gradniki strani. Najboljša praksa določa, da se zunanje .js datoteke navede na koncu in ne v glavi ($<head>$), saj tako manj ovirajo nalaganje strani.

Koda 3.1: Struktura dokumenta HTML.

```
<!DOCTYPE html>
<html>
<head>
    <title>Naslov zavihka</title>
    <link rel="stylesheet" type="text/css" href="
       style.css">
</head>
<body>
    <h1>Naslov</h1>
    <p>Odstavek</p>
</body>
</html>
```

3.2 CSS

Cascading Style Sheets (kaskadne stilske predloge) je jezik, ki se ga uporablja za definiranje izgleda spletnih strani. Z njim opisujemo predstavitev dokumenta napisanega v HTML, skrbimo za postavitev elementov, poravnave,

odmike ter tudi za izgled posameznih elementov. Uporablja se tudi za animacije. Bistvo jezika CSS je, da ločimo strukturo (HTML) in izgled strani. Pri določanju množice elementov, nad katerimi bomo uporabili določene stilske predloge, uporabljamo različne selektorje:

- značka HTML - sem spadajo vse značke HTML, npr **div, p, a**,
- razredi - razrede se označi s ., npr **.wrapper**,
- identifikator ali id - označi se jih s #, npr **#main**,
- psevdo razredi - določajo posebno stanje elementa, označimo jih z **:**, npr **a:hover**,
- psevdo elementi - uporabljajo se za določitev stilskih predlog za določene dele elementa. Označujemo jih z **::**, npr **p::first-line**.

Te selektorje se lahko med sabo tudi združuje, npr **#main .wrapper**. Tak selektor bo najprej poiskal, kje se v dokumentu nahaja **#main**, nato bo v tem selektorju poiskal **.wrapper** in mu dodal stilsko predlogo. Primeri uporabe selektorjev za določitev stila elementov so prikazani v kodi 3.2.

Koda 3.2: Struktura CSS.

```
.wrapper{
    background: red !important;
}
.wrapper h1{
    color: black;
}
.wrapper p::first-line {
    color: #ff0000;
}
```

Vsak selektor ima različno težo. V primeru, ko imamo za nek element več različnih selektorjev, pa selektor oziroma kombinacija selektorjev z višjo

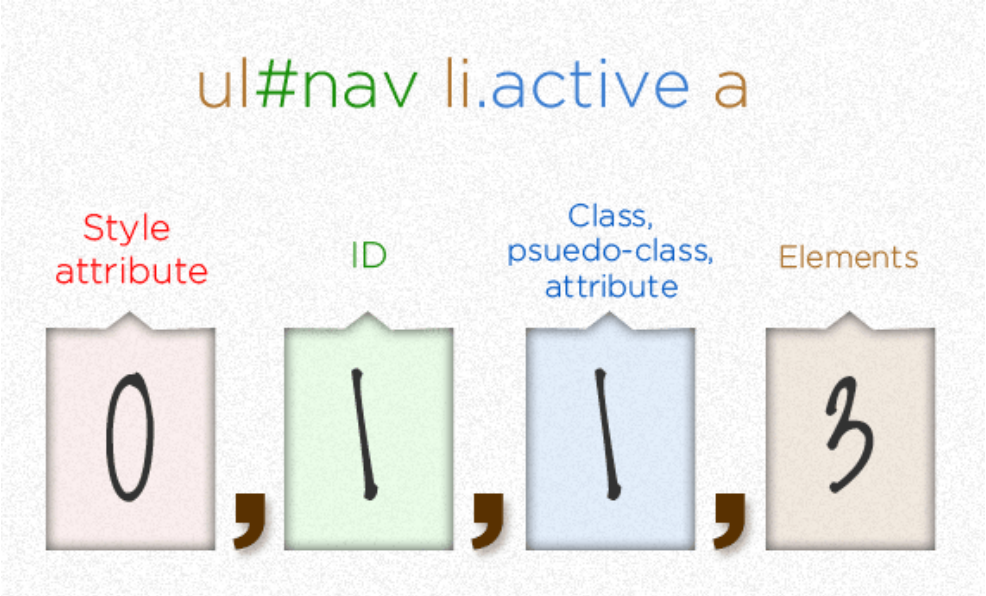
selektor	css znak	teža
značka HTML	div, h1, p, ...	1
razred	.	10
identifikator	#	100
attribut style v html		1000

Tabela 3.1: CSS: Teže selektorjev.

težo prevlada. Teže posameznih selektorjev so prikazane v tabeli 3.1.

Če vsebuje vrednost lastnosti selektorja ključno besedo **!important**(koda 3.2), potem ta lastnost prevlada ne glede na težo.

Slika 3.1 prikazuje računanje teže selektorjev. Imamo tri značke HTML (ul, li in a), en razred (.active) ter en identifikator (#nav) kar pomeni, da je skupna teža 113.



Slika 3.1: Primer računanja teže selektorjev CSS [2].

3.2.1 SASS/SCSS

SASS (Syntactically awesome stylesheets) [3] je izboljšana oblika jezika CSS. Omogoča nam uporabo funkcij, uvoz datotek, uporabo spremenljivk in gnezdenje CSS ukazov. Sassy CSS (SCSS) je novejša sintaksa jezika SASS. Brskalniki SCSS oblike zapisa ne prepoznajo, zato je potrebno kodo SCSS pretvoriti v navadno obliko CSS. Koda 3.3 prikazuje gnezdeno obliko sintakse SCSS, ki se nato pretvori v zapis prikazan v kodi 3.2.

Koda 3.3: Gnezdena struktura SCSS.

```
.wrapper{
    background: red !important;
    h1{
        color: black;
    }
    p::first-line {
        color: #ff0000;
    }
}
```

3.3 JavaScript

Je skriptni programski jezik, ki se uporablja za izdelavo dinamičnih vsebin in spreminjanje spletne strani na strani odjemalca. JavaScript se lahko neposredno vključi v HTML ali pa preko povezave na JavaScript datoteko. Tipične uporabe JavaScripta so validacija vnosnih polj, branje in pisanje piškotov, ravnanje z dogodki (ang. event handling), kot so klik na gumb, sprememba vnosnega polja, itd. JavaScript je v naboru spletnih tehnologij dandanes nepogrešljiv člen, s katerim so razvita številna močna ogrodja in knjižnice, ki omogočajo lažje delo in prinašajo številne nove možnosti razvoja.

Nekaj ogrodij in knjižnic razvitih z JavaScript:

- jQuery - knjižnica za poenostavitev upravljanja s HTML na odjemalcu,
- React - knjižnica za izdelavo uporabniških vmesnikov,
- Angular.js/Angular - ogrodje/platforma za izdelovanje mobilnih in spletnih aplikacij,
- Node.js - izvajalno okolje za izdelavo orodij in aplikacij,
- npm - upravitelj paketov za JavaScript,
- Vue.js - ogrodje za izdelovanje interaktivnih vmesnikov,
- Ionic - SDK (software development kit) za izdelavo izvornih (ang. native) aplikacij.

3.3.1 TypeScript

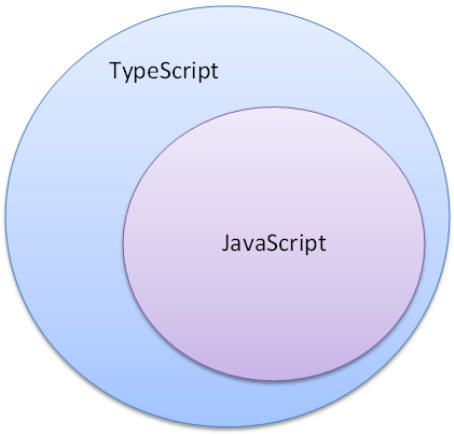
TypeScript je odprtokodni programski jezik, ki ga je razvil Microsoft. Je nadgradnja jezika JavaScript, namenjena predvsem srednjim in velikim aplikacijam. TypeScript vsebuje veliko novih funkcij, ki jih JavaScript trenutno še ne vsebuje. Razlog za to je predvsem podpora brskalnikov, saj mnogi brskalniki še nimajo podpore za številne dodatne funkcionalnosti.

Funkcionalnosti dodane v TypeScript [4]:

- razredi,
- moduli,
- funkcije lambda,
- tipi spremenljivk,
- vmesniki,
- tipi vmesnikov,
- tip enum,

- async/ await,
- generično programiranje,
- terke (ang. tuple).

Ker brskalniki ne razumejo TypeScript, ga moramo najprej pretvoriti v JavaScript. Ta proces imenujemo **transpiling**, kjer se TypeScript pretvori v primerno kodo JavaScript, ki jo brskalniki razumejo. Vsaka veljavna koda JavaScript se upošteva kot veljaven TypeScript. Slika 3.2 simbolično predstavlja nabor funkcionalnosti, ki jih JavaScript še ne vsebuje.



Slika 3.2: Prikaz nabora funkcionalnsti jezikov JavaScript in TypeScript [5].

3.4 JSON

JSON (JavaScript object notation) je preprost datotečni format zapisan v berljivi tekstovni obliki. Je jezikovno neodvisen format, ki izhaja iz jezika JavaScript. Zaradi njegove preprostosti in razloga, da si odjemalec in strežnik lahko pošiljata podatke samo v tekstovnem formatu, je najbolj uporabljen format za pošiljanje podatkov med odjemalcem (brskalnik) in strežnikom. JSON vedno bolj izpodriva format XML (Extensible Markup Language), ki

je bil osnovni format pri komunikaciji Ajax (asynchronous JavaScript and XML). Osnovni strukturi formata JSON sta objekt (zbirka parov ključ-vrednost) in polje (ang. array).

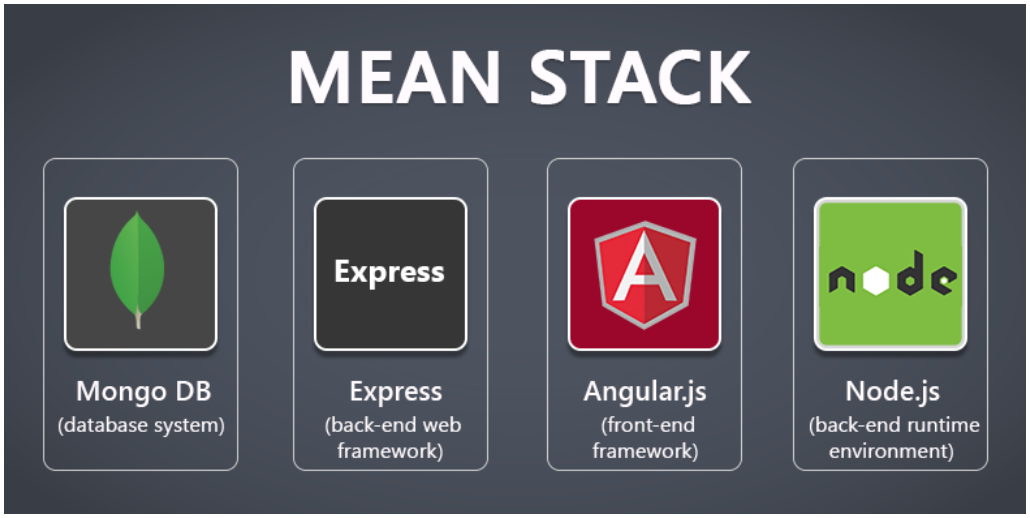
Koda 3.4 predstavlja primer polja objektov. V njej se nahajajo trije objekti, ki vsebujejo po dva para ključ-vrednost.

Koda 3.4: Primer zapisa podatkov v formatu JSON, sestavljenega iz polja treh objektov.

```
[
    {
        "sadez": "Banana",
        "barva": "rumena"
    },
    {
        "sadez": "Jabolko",
        "barva": "rdeca"
    },
    {
        "sadez": "Hruska",
        "barva": "zelena"
    }
]
```

3.5 MEAN

MEAN je oznaka za brezplačno in odprtokodno zbirko tehnologij, ki temeljijo na programskem jeziku JavaScript in se uporabljajo za izdelavo dinamičnih spletnih strani in spletnih aplikacij. V to zbirko spadajo MongoDB, Express.js, Angular in Node.js. Obsega bazo podatkov (MongoDB), ogrodje (Express), ki teče na strežniku, ogrodje (Angular), ki se izvaja na odjemalcu (ang. client-side) in izvajalno okolje (Node.js), ki teče na strežniku (ang. server-side).



Slika 3.3: Prikaz nabora tehnologij v MEAN stack [6].

3.5.1 MongoDB

MongoDB je odprtokodna, NoSQL (not only SQL) podatkovna baza, ki podatkov, za razliko od baz SQL, ne shranjuje v tabele, pač pa v zbirke (collections) dokumentov. Ti dokumenti so zapisani v formatu JSON. Namen takega shranjevanja je hiter prenos podatkov med odjemalcem in stražnikom. Dokumenti MongoDB so razširljivi, saj nimajo popolnoma določene sheme za shranjevanje. To pomeni, da sta lahko dva dokumenta v isti zbirki popolnoma različna, saj shema dokumenta ni nujno potrebna, je pa vseeno priporočeno uporabljati sheme za enolično shranjevanje dokumentov v zbirke podatkov. Dokumente se lahko, prav tako kot pri tabelah SQL zapise, indeksira, kar pomeni, da se, ko poženemo poizvedbo na zbirki podatkov, dokumente po določenem ključu hitreje najde. Privzeto so unikatni identifikatorji dokumenta tudi indeksi, lahko pa določimo lastne indekse, vendar to vpliva na hitrost poizvedb pri zapisovanju v bazo, saj se morajo indeksi ponovno nastaviti.

3.5.2 Express

Express je ogrodje (ang. framework), ki nam pomaga pri grajenju spletnih aplikacij. Deluje na različnih operacijskih sistemih in je odprtokoden. Zagotavlja veliko število uporabnih in močnih funkcij (ang. features), ki olajšajo izdelavo spletnih aplikacij. Skrbi, da nam ni potrebno ponovno programirati že vpeljanih in razvitih metod (usmerjanje, postavitev okolja na strežniku), zato se lahko osredotočimo na samo aplikacijo. Dve najpomembnejši funkciji, ki jih zagotavlja, sta usmerjanje (ang. routing) in obravnavanje zahtev (ang. request handling). Skrbi tudi za komunikacijo s podatkovno bazo. Namenjen je postavitvi spletnega strežnika v Node.js izvajalnem okolju.

3.5.3 Angular

Angular (Angular2) je JavaScript ogrodje namenjeno spletnim, namiznim in mobilnim aplikacijam. Razvilo ga je podjetje Google. Izšlo je 14. septembra leta 2016. Je naslednik zelo popularnega ogrodja Angular.js, vendar ni samo nadgradnja Angular.js, ampak popolnoma predelana nova različica.

Ključne razlike med Angular in Angular.js:

- modularnost - veliko funkcionalnosti se je preselilo v module, kar omogoča hitrejše delovanje jedra.
- uvedba komponent - aplikacija se deli na številne komponente. Komponente so deli aplikacije, ki vsebujejo predloge (template/HTML) in logiko za pravilno delovanje komponent.
- razvoj za mobilne naprave - pri razvoju ogrodja je bila pozornost posvečena tudi mobilnim napravam. Omogoča razvoj izvornih (ang. native) mobilnih aplikacij.
- grajenje aplikacij, ki se jih lahko namesti na osebni računalnik (ne glede na operacijski sistem).

- izboljšana hitrost in delovanje.
- asinhrono prevajanje predlog.
- poenostavljeno preusmerjanje (ang. routing).
- reaktivno programiranje.

Naloga Angular je, da uporabniku omogoči čim boljšo uporabniško izkušnjo. Tako ni potrebno pošiljati zahtev in čakati na odgovor, vse se dogaja v realnem času.

3.5.4 Node.js

Node.js je platforma, ki temelji na Chromovem JavaScript izvajalnem okolju (ang. Chrome's JavaScript runtime). Namenjen je gradnji hitrih in razširljivih spletnih aplikacij. Node.js se izvaja na strežniku, za razliko od JavaScript, ki deluje v brskalniku. Omogoča funkcionalnosti, kot so dostop do datotečnega sistema, poslušanje prometa na omrežju, dostopanje do podatkovne baze in poslušanje na zahteve HTTP ter pošiljanje odgovorov HTTP (ang. HTTP handling). Node.js aplikacije so napisane v JavaScriptu.

3.6 Webpack

Webpack je orodje, ki iz vseh naših virov (JavaScript, CSS, slike, pisave) naredi graf odvisnosti. S pomočjo tega grafa združi večje število modulov v eno samo datoteko. Webpack v osnovi procesira samo datoteke JavaScript. V primeru ko hočemo procesirati še druge vrste datotek, moramo vključiti nalagalnike (ang. loaders). Na ta način lahko ostale vire pretvorimo v module JavaScript. Za dodatne funkcionalnosti, kot je minifikacija, pa moramo vključiti dodatne vtičnike (ang. plugins).

Namen Webpacka:

- razdeliti drevo odvisnosti na manjše dele celote (chunk),
- zmanjšati čas nalaganja spletnih strani - naložimo le del celote, ostale dele naložimo po potrebi,
- združiti več datotek v eno.

Poglavje 4

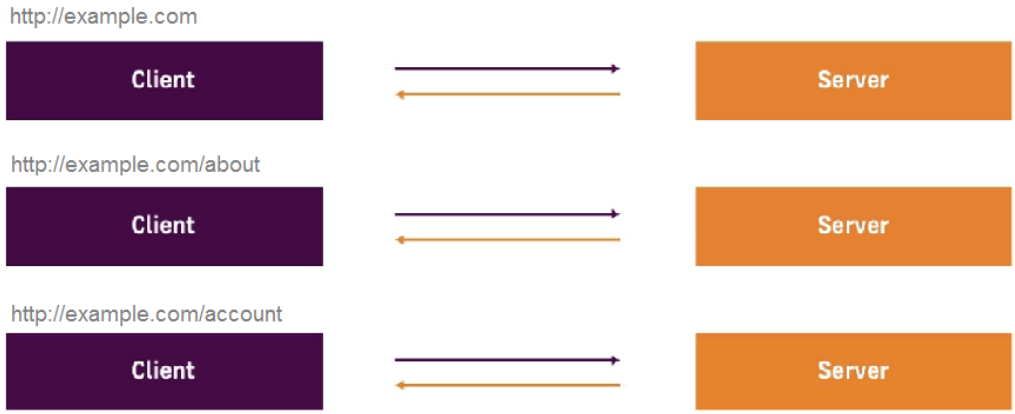
Angular terminologija

V tem poglavju so bolj podrobno opisani glavni gradniki Angular aplikacije [7], kot so komponente, moduli, servisi, filtri in usmerjevalniki ter zaščita poti.

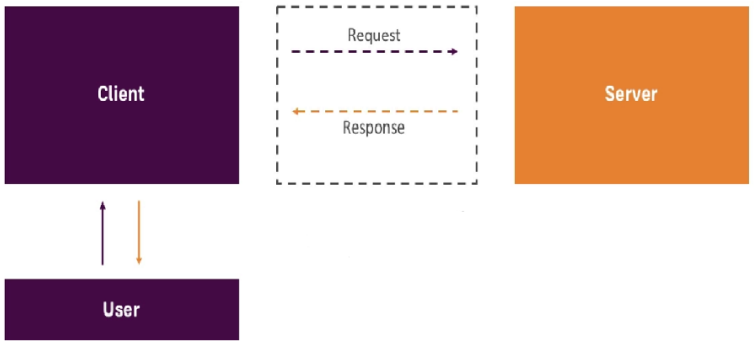
4.1 SPA

Angular aplikacije spadajo v kategorijo enostranskih aplikacij (SPA, single page application). Enostranske aplikacije so aplikacije, ki naložijo eno samo stran HTML in to stran potem dinamično posodabljajo glede na uporabnikovo interakcijo. To pomeni, da se vse dogaja na strani odjemalca(brskalnika), kar omogoča hitrejše delovanje, saj se zahteve ter odgovori med strežnikom in odjemalcem ne pošiljajo, to pa omogoča nemoteno delovanje brez čakanja na končanje komunikacije. Tradicionalne spletne strani pošiljajo zahteve na strežnik vedno, ko zamenjamo URL (slika 4.1). SPA aplikacije pa to storijo samo ob prvem nalaganju strani, za ostalo preusmerjanje pa poskrbi JavaScript (slika 4.2) in s tem uporabniku da občutek hitrega delovanja. Pogovor med odjemalcem in strežnikom se dogaja samo, ko moramo pridobiti podatke (dostop do baze podatkov) ali če uporabljamo leno nalaganje (ang. lazy loading). Aplikacije zgrajene na tak način so veliko hitrejše, saj ni potrebno pošiljati podatkov preko interneta, s tem pa tudi ni potrebno čakati

na odgovor strežnika. Vse se dogaja v brskalniku. Cilj takih aplikacij je, da uporabniku omogočijo podobno izkušnjo kot pri aplikacijah nameščenih na računalniku.



Slika 4.1: Zahteve in odgovori pri tradicionalnih spletnih straneh (z dovoljenjem avtorja Maximiliana Schwarzmüllerja).



Slika 4.2: Zahteve in odgovori pri SPA (z dovoljenjem avtorja Maximiliana Schwarzmüllerja).

4.2 Leno nalaganje

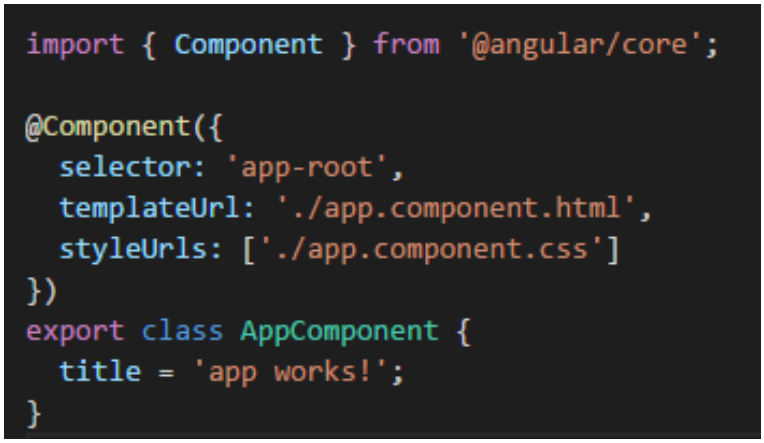
Leno nalaganje oziroma nalaganje po potrebi je pojem, ki opisuje dogodek, ko potrebno informacijo naložimo takrat, ko jo potrebujemo. Tak način omogoča bolj učinkovito in hitrejše delovanje aplikacije. Mi smo uporabili leno nalaganje, ko se zamenja URL. Ko uporabnik prvič preide na novo stran, se naloži del aplikacije, ki je zadolžen za delovanje tega dela aplikacije. Če je uporabnik do tega dela že dostopal, se ta del ne naloži ponovno, saj je bil že naložen in ni potrebe po ponovnem nalaganju. Dobra stran tega načina je, da je začetno nalaganje strani hitrejše, saj pridobimo le podatke, ki jih potrebujemo. Slaba plat pa se vidi, če ima uporabnik slabo internetno povezavo in mora počakati, da se pridobijo podatki s strani strežnika (to daje podoben občutek nalaganja kot pri tradicionalnih spletnih straneh).

4.3 Komponente

HTML prihaja s številnimi vgrajenimi značkami, kot so , <p>, <a>, ki imajo svoj izgled in obnašanje. Pri Angular aplikacijah pa naredimo nove značke, ki imajo svoj izgled in se obnašajo, kot jim določimo. Take značke imenujemo komponente. Komponente so glavni del Angular aplikacije. Angular aplikacija pa je skupek komponent, ki vzajemno delujejo.
Komponente so sestavljene iz treh delov delov (uvožene metode, meta podatki in razred), kar prikazuje slika 4.3.

4.3.1 Uvožene metode

Pod uvožene metode (ang. imports) spadajo odvisnosti, ki so definirane v drugih datotekah. Z uvozom omogočimo uporabo teh odvisnosti v naših komponentah. Pod uvožene odvisnosti spadajo razredi, vmesniki (ang. interface), servisi in drugo. Uvoz odvisnosti se ne uporablja samo pri komponentah, potrebujemo jih tudi pri modulih, servisih, filtrih, razredih.



Slika 4.3: Prikaz sestavnih delov Angular komponente: uvožene metode (import), meta podatki (@Component) in razred (class).

4.3.2 Meta podatki

Meta podatki so podatki, ki definirajo našo komponento. Pri komponentah se za meta podatek uporablja označevanje *@Component*. Med te podatke spadajo *selector*, *template* ali *templateUrl*, *styles* ali *styleUrls* in še mnoge druge. Izmed vseh sta najbolj pomembna *selector* in *template/templateUrl*. S selektorjem določimo, pod kakšno značko HTML bo naša komponenta dostopna. Na sliki 4.3 je pod *selektor* določena vrednost **app-root**, kar nakazuje na to, da bo naša komponenta dostopna pod značko *<app-root></app-root>*, kar je prikazano v kodi 4.1. Selektorji morajo biti unikatni, da komponente ne povzročajo konfliktov med sabo. Najboljša praksa je, da selektorjem dodamo predpono (na sliki 4.3 je določena predpona *app*).

Koda 4.1: Uporaba komponente v HTML.

```
<body>
    <app-root></app-root>
</body>
```

V *template* vnesemo HTML, katerega želimo, da ga naša komponenta vsebuje, medtem ko *templateUrl* kaže na datoteko HTML, v kateri je HTML

koda. Najboljša praksa veleva, da se uporablja *templateUrl*, saj je s tem naša koda bolj berljiva, bolje strukturirana ter bolj čista. To še posebej velja za komponente, ki vsebujejo veliko HTML-ja.

4.3.3 Razred

Pod razred spada vsa logika naše komponente. V njem se določa lastnosti komponente, povezovanje s servisi, lovljenje dogodkov (na primer klik na gumb) in aritmetične operacije. Ime razreda mora biti unikatno definirano, da ne povzroča konfliktov. Dobra praksa je, da na koncu imena dodamo besedo **Component**, da vemo, da ta razred pripada komponenti. Prav tako moramo dodati besedo **export**, če želimo, da je komponenta dostopna drugim modulom.

4.4 Moduli

Pri Angular aplikacijah je koda strukturirana v pakete, ki jih imenujemo moduli ali **NgModule**. Vsaka Angular aplikacija potrebuje vsaj en modul, ki se ga običajno poimenuje *AppModule*. Ta modul je koren (ang. root) aplikacije, na katerega se sklicujejo ostali deli aplikacije. V JavaScriptu se izraz *module* navezuje na kodo v eni sami datoteki. **NgModule** pa ima malo drugačen koncept. **NgModule** združuje kodo iz različnih datotek v pakete. **NgModule** vsebuje funkcionalnosti (komponente, servise, filtre in direktive) iz različnih datotek v eni sami datoteki. Vsebovane funkcionalnosti so tako dostopne v vseh gradnikih Angular aplikacije, ki jih ta modul vsebuje. S takim načinom nam ni potrebno uvažati funkcionalnosti v vsak gradnik posebej, ampak so zbrane na enem mestu (modulu).

Kot je razvidno iz slike 4.3 in 4.4, sta si koncepta zelo podobna. Oba imata na vrhu uvožene razne funkcionalnosti, nato sledijo meta podatki in nato razred. Tak pristop bo viden tudi pri ostalih Angular gradnikih, kot so servisi in filtri. Module (slika 4.4) označujemo z izrazom **NgModule**. Dobra

praksa narekuje, da se različne funkcionalnosti aplikacije deli v svoje module, tako se na primer deli funkcionalnosti, ki imajo povezavo z uporabnikom (ang. User) in funkcionalnosti v zvezi s plačili (ang. Payment) deli na različne module (primer *UserModule* in *PaymentModule*). S takim načinom naredimo kodo bolj organizirano in pregledno.

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { HttpModule } from '@angular/http';

import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpModule
  ],
  exports: [],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Slika 4.4: Angular modul.

Nekaj najbolj pomembnih parametrov, ki jih sprejme **NgModule**:

- **imports** - ta parameter oziroma tabela parametrov vsebuje module, ki jih želimo uporabljati v tem modulu in komponentah, do katerih ima modul dostop.
- **declarations** - pod ta parameter spadajo komponente, filtri in direk-

tive (ang. directives). Vse, kar je bilo deklarirano v modulu, se ne sme ponovno deklarirati v drugih modulih, saj to povzroči napako v kodi. Če želimo iste komponente uporabljati v različnih modulih, je najbolje, da se naredi skupni modul, ki ga vključimo pod *imports*.

- **exports** - sem spadajo komponente, filtri in direktive, za katere želimo, da so dostopne izven modula, v katerem so deklarirane. Če gradniki niso dodani v to tabelo in če modul ni dodan pod imports tabelo v drugem modulu, ti gradniki ne bodo dostopni v drugih modulih.

- **providers** - pod providers tabelo spadajo servisi. Ti servisi bodo dostopni pri vseh gradnikih, ki jih ta modul vsebuje.

- **bootstrap** - je parameter, ki je dostopen samo pri glavnem modulu (ang. root). Ta parameter določa, s katero komponento se bo naložila aplikacija.

Angular že v osnovi prihaja s številnimi vgrajenimi moduli, ki jih lahko po želji vključimo. Nekateri najpogostejši so:

- **BrowserModule** - registrira pomembne servise za delovanje v brskalniku, vsebuje nekaj ključnih direktiv (NgIf, NgFor), ki so po vključitvi dostopne v naših komponentah.

- **CommonModule** - BrowserModule se lahko uporablja samo v glavnem modulu, zato CommonModule nudi storitve, kot so direktive NgIf in NgFor in nam omogoči uporabo le teh v drugih modulih. BrowserModule že v osnovi vsebuje CommonModule, zato ni potrebno dodati obeh.

- **FormsModule** - omogoča funkcije povezane z obrazci.

- **HttpModule** - funkcije HTTP, kot so GET in POST zahteve.

- **RouterModule** - metode, ki se nanašajo na usmerjanje (ang. routing).

4.5 Servisi

Servisi (ang. services) so delno namenjeni temu, da podvojeno kodo, ki se nahaja na večih mestih, premaknemo na eno mesto. S tem sledimo principu DRY (ang. don't repeat yourself), ki predpisuje, da naj se koda ne podvaja. Servisi poskrbijo za to, da se koda, ki se v kodi pojavlja večkrat, nahaja na enem mestu.
Nekateri najbolj pogosti primeri uporabe servisov so:

- interakcija s strežnikom - pod to rubriko spadata zapisovanje podatkov v bazo in pridobivanje podatkov s strežnika. Praktično obsega vso komunikacijo s strežnikom.

- komunikacija med komponentami/razredi - pošiljanje in lovljenje dogodkov. Primer: Prva komponenta pošlje podatke o dogodku, druga komponenta pa te podatke potrebuje. Najboljši način je, da prva komponenta pošlje podatke v servis, druga komponenta pa posluša na spremembo podatkov v servisu.

- dostop do istih podatkov - več različnih komponent potrebuje podatke. Najboljše je, da so podatki dostopni iz enega mesta (servisa).

Slika 4.5 predstavlja servis, ki nam vrne vsoto dveh števil. Na ta način lahko različne komponente dostopajo do iste metode, zato ni potrebno v vsaki komponenti posebej podvajati kode, prav tako pa se tudi izognemo popravljanju kode na večih mestih.

4.6 Filtri

Filtri so zadolženi za vizualno pretvorbo podatkov, kar pomeni, da se spremeni zgolj vizualni prikaz podatkov. Ta pretvorba vpliva samo na podatke, ki jih vidijo uporabniki. V ozadju ti podatki niso spremenjeni. Ena izmed

```
import { Injectable } from '@angular/core';

@Injectable()
export class MathService {

  constructor() { }

  getSum(a: number, b: number){
    return a + b;
  }
}
```

Slika 4.5: Angular servis za seštevanje števil.

mnogih uporab filtrov je, da besedilo, ki je napisano z malimi črkami, pretvorimo v besedilo, ki je napisano z velikimi črkami.
Angular prihaja z vnaprej vgrajenimi filtri, kot so:

- **uppercase** - besedilo z velikimi črkami,
- **lowercase** - besedilo z malimi črkami,
- **date** - spremeni datum na določen format,
- **currency** - številko pretvori v ceno z valuto,
- **percent** - številko pretvori v procente.

Slika 4.6 prikazuje uporabo filtrov v predlogah komponente. Filtre se prepozna po znaku |, ki predstavlja ukaz za preusmeritev toka podatkov. Vse, kar je pred znakom |, je vrednost, ki jo želimo spremeniti, kar pa je za znakom |, pa so filtri, ki jih želimo uporabiti nad vrednostjo. Koda na sliki 4.6 prikazuje primer, ko želimo vrednost *test value* spremeniti v male začetnice, v drugem odstavku pa v velike začetnice. V prvem odstavku bo rezultat pretvorbe *test value*, medtem ko bo v drugem odstavku rezultat *TEST VALUE*.

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'lowerupper-pipe',
  template: `<div>
    <p>In lowercase: <pre>'{{value | lowercase}}'</pre>
    <p>In uppercase: <pre>'{{value | uppercase}}'</pre>
  </div>`
})
export class LowerUpperPipeComponent {
  value: string = "Test value";
}
```

Slika 4.6: Uporaba filtrov v komponentah.

4.7 Direktive

Direktive (ang. directives) so navodila, ki Angularju povejo, kaj naj naredi na določenih delih kode. Direktive se uporablja kot značke HTML ali kot atribute na značkah HTML. Zato komponente spadajo pod rubriko direktiv, le da imajo dodano še predlogo (ang. template).

Direktive delimo na dve vrsti:

- **atributne** - to so direktive, ki se značkam HTML dodajo kot atribut, po tem tudi ime atributne direktive. Te direktive ne spreminjajo strukture dokumenta. Vplivajo pa lahko na izgled elementa. Dve izmed najbolj prepoznavnih atributnih direktiv sta **NgClass** in **NgStyle**. Direktiva **NgClass** elementu doda ali odvzame razred (selektor CSS) ob določenem pogoju, medtem ko **NgStyle** spremeni CSS lastnosti elementa.

  Slika 4.7 prikazuje primer uporabe atributnih direktiv. Pri **NgClass** (selektor *[ngClass]*) bomo HTML elementu dodali razred *active*, če bo

```
<div [ngClass]="{'active': value == 1}"></div>

<div [ngStyle]="{'background': background_color}"></div>
```

Slika 4.7: Atributne direktive nad HTML elementom.

vrednost lastnosti *value* enaka 1. Pri **NgStyle** (selektor *[ngStyle]*) pa bomo elementu HTML dodali barvo ozadja glede na vrednost lastnosti *background_color*.

- **strukturne** - strukturne direktive pa vplivajo tudi na strukturo dokumenta HTML. Najbolj uporabljeni strukturni direktivi, ki sta že vgrajeni v Angular, sta **NgIf** in **NgFor**. Strukturna direktiva **NgIf** iz HTML dokumenta izbriše element in njegove naslednike (element izbriše in ne samo skrije), medtem ko **NgFor** iterira čez tabelo elementov in vsak element doda v HTML.

  Slika 4.8 prikazuje primer uporabe strukturnih direktiv nad elementi HTML. Direktiva **NgIf** (selektor **ngIf*) bo iz dokumenta HTML izbrisala element z razredom *element*. Direktiva **NgFor**(selektor **ngFor*) pa bo za vsak element v tabeli *items* naredila novo točko (<li>) v seznamu (<ul>). Iz slike je razvidno, da bo seznam imel tri elemente, saj so v tabeli *items* trije zapisi, vsaka točka pa bo vsebovala besedilo določeno v iteraciji čez tabelo.

```
<div class="element" *ngIf="value == 1">
  If value is 1 delete element
</div>

<!-- items = ["first","second","third"] -->
<ul>
  <li *ngFor="let item of items">{{item}}</li>
</ul>
```

Slika 4.8: Strukturne direktive nad dokumenta HTML.

Poglavje 5

Razvoj

V tem poglavju je opis implementacije spletne storitve. Najprej je opisana implementacija funkcionalnih zahtev uporabnika, sledi opis razvoja plačilnega sistema, na koncu je še predstavljen uporabniški vmesnik za grajenje spletnih strani in razvoj le tega.

5.1 Usmerjanje

Za enostranske aplikacije je pomembno, da se pravilno določijo prehajanja med posameznimi vsebinami prikazanimi na strani. Za to poskrbi usmerjevalnik (ang. router). Usmerjevalnik posluša na spremembe, ki se zgodijo z naslovom URL in izvede potrebno akcijo. Slika 5.1 prikazuje glavni navigacijski meni za našo aplikacijo. Preko tega navigacijskega menija lahko uporabnik spreminja URL strani. Pri tradicionalnih aplikacijah bi se zgodila preusmeritev, pri enostranskih aplikacijah pa se zgolj zamenja vsebina trenutne strani, brez osveževanja ali preusmeritve. Ko usmerjevalnik dobi obvestilo o spremembi spletnega naslova, pogleda v usmerjevalno tabelo in naloži komponento, ki ustreza zahtevi.

Slika 5.2 prikazuje usmerjevalno tabelo (polje objektov), ki predstavljajo posamezne poti. Vsak objekt je sestavljen iz poti (*path*) in komponente (*component*), ki jo je potrebno naložiti. V primeru, da objekt poti vsebuje

Slika 5.1: Glava spletne strani.

lastnost *loadChildren*, se bo preko lenega nalaganja naložil modul, ki ga zahtevamo. V tem modulu se nahaja nova usmerjevalna tabela, ki ima dodatne podatke o usmerjanju in komponentah, ki se bodo izrisale.

```
const routes: Routes = [
    { path: '', pathMatch: 'full', component: LandingComponent },
    {
        path: 'editor',
        loadChildren: './editor/editor.module#EditorModule',
        canActivate: [AuthGuard]
    },
    {
        path: 'market',
        loadChildren: './subscription/market.module#MarketModule'
    },
    {
        path: 'auth',
        loadChildren: './auth/auth.module#AuthModule'
    },
    // redirect unmathched routes to error page
    { path: '**', component: PageNotFoundComponent }
];

export const routing = RouterModule.forRoot(routes);
```

Slika 5.2: Glavni usmerjevalnik aplikacije.

Ker enostranske aplikacije ne preusmerjajo strani, se zahtevane komponente izrišejo tam, kjer naš dokument HTML vsebuje *<router-outlet></router-outlet>* (koda 5.1).

Koda 5.1: Mesto, kamor se bodo izrisale komponente ob usmerjanju.

```
<main>
    <router-outlet></router-outlet>
```

```
</main>
```

5.2 Uporabniki

Spletna aplikacija je namenjena grajenju lastnih spletnih strani preko brskalnika, zato je nujno potrebna implementacija uporabnikov. Uporabniki bodo imeli unikatno določen identifikator, ki bo določal, komu bodo pripadale rešitve, ki so bile narejene preko uporabniškega vmesnika. Slika 5.3 predstavlja MongoDB shemo [8] uporabnika. To so podatki, ki se shranjujejo v podatkovno bazo. Lastnosti *firstName* in *lastName* predstavljata ime in priimek uporabnika, lastnost *password* je geslo uporabnika, lastnost *email* predstavlja elektronski naslov uporabnika in hkrati tudi njegovo uporabniško ime, lastnost *allowed_solutions* predstavlja število rešitev, ki jih uporabnik lahko naredi, medtem ko je lastnost *role* vloga uporabnika v sistemu. Na voljo sta dve vlogi:

- **uporabnik** - vloga predstavlja navadne uporabnike, ki imajo osnovne pravice, kot sta kreiranje rešitev in kupovanje planov,
- **administrator** - ima osnovne pravice in dostop do nadzorne plošče.

```
var schema = new Schema({
    firstName: { type: String, required: true },
    lastName: { type: String, required: true },
    password: { type: String, required: true },
    email: { type: String, required: true, unique: true },
    role: { type: Number },
    allowed_solutions: { type: Number }
});
```

Slika 5.3: MongoDB shema uporabnika.

5.2.1 Registracija uporabnika

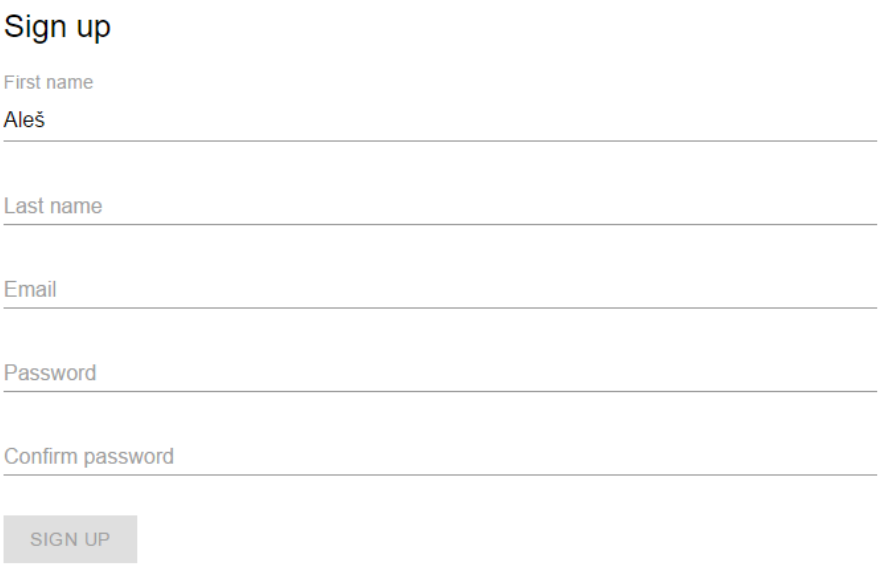
Ker brez uporabnikov storitev ne mora zaživeti, je bilo potrebno razviti registracijo, kjer bodo uporabniki ustvarili svoj uporabniški račun. Slika 5.4 prikazuje obrazec za registracijo s polji za podatke,, ki jih mora uporabnik vnesti za uspešno registracijo. Ko so vsi podatki vnešeni in ustrezajo vsem pogojem, se sprosti gumb za registracijo. Pogoji za registracijo so:

- **obvezna polja** - vsa polja obrazca so obvezna, zato jih mora uporabnik izpolniti,
- **veljaven elektronski naslov** - elektronski naslov mora ustrezati pravilu za veljavne elektronske naslove,
- **geslo** - geslo mora biti dolgo vsaj šest znakov,
- **gesli morata biti enaki** - geslo in geslo za potrditev se morata ujemati.

Ko so vsi podatki vnešeni in ustrezajo vsem pogojem, se sprosti gumb (obarva zeleno) in omogoči registracijo. Pri registraciji se geslo zakodira, kar omogoči večjo varnost. V primeru, da se ob registraciji zgodi napaka, se na vrhu obrazca prikaže obvestilo v obliki rdečega pravokotnika, v katerem se z belo barvo izpiše napaka. Edina napaka, ki se lahko pripeti med registracijo je, da uporabnik z izbranim elektronskim naslovom že obstaja.

5.2.2 Prijava

Obrazec za prijavo, ki je prikazan na sliki 5.5, je precej podoben obrazcu za registracijo. Edina razlika je, da je za prijavo potrebno vnesti elektronski naslov (uporabniško ime) in geslo, ter da je za sprostitev gumba za prijavo dovolj, če uporabnik vnese zahtevana podatka. Pri prijavi se lahko uporabniku prikažejo napake, kot so nepravilno geslo ali če uporabnik ne obstaja (nima ustvarjenega uporabniškega računa). Obvestila o napakah se prikažejo na enak način kot pri registraciji.



Slika 5.4: Obrazec za registracijo uporabnikov.

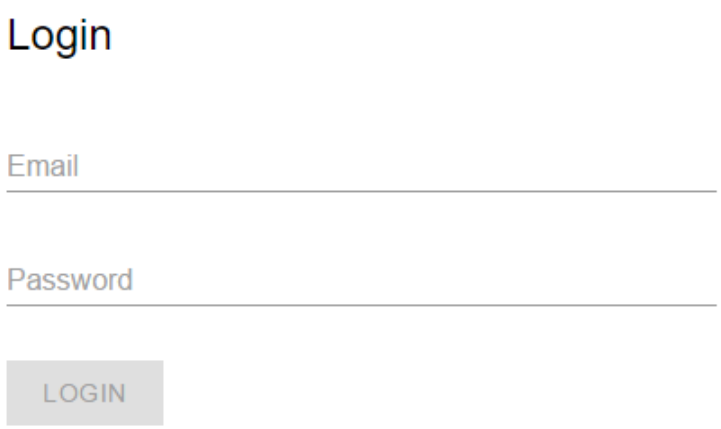
Ob prijavi uporabnika se zamenja tudi navigacijski meni v glavi. Odstranita se gumba za registracijo in prijavo, spremeni se gumb *Pricing* (seznam paketov) v *Market*(nakup), doda pa se gumb do profila uporabnika ter gumb za odjavo.

5.2.3 Odjava

Ko se uporabnik odjavi, ga preusmerimo na začetno stran in spremenimo navigacijski meni v prvotno stanje (slika 5.1).

5.3 Plačilni sistem

Vsak servis, ki uporabnikom nudi določene storitve, mora imeti implementiran plačilni sistem. Mi nismo razvijali svojega sistema, ampak smo uporabili Stripe [9]. Stripe je platforma, ki omogoča plačevanje preko interneta. Uporabniku smo omogočili izbiro med tremi paketi, ki za različno ceno omogočajo določeno število rešitev (spletnih strani). Uporabnik lahko te zakupljene



Slika 5.5: Obrazec za prijavo uporabnika.

rešitve uporabi za ustvarjanje spletnih strani. Ko uporabnik izbere plan, se mu prikaže plačilni obrazec, kamor vnese elektronski naslov, na katerega se mu pošlje račun, številko kartice, datum veljavnosti kartice in verifikacijsko šifro kartice. Ko uporabnik potrdi nakup, se preko Stripe API-ja (ang. application programming interface) opravi plačilo. Za uporabo Stripe API-ja je na njihovi spletni strani potrebno narediti račun in pridobiti skrivni in javno dostopni ključ. Skrivni ključ se uporabi na strani strežnika za izvedbo plačila, medtem ko je javni ključ uporabljen za pretvorbo podatkov o kartici v žeton (ang. token) [10]. Žeton lahko pridobimo s Stripovo knjižnico checkout.js, ki podatke o kartici kot zahtevo (ang. request) pošlje na Stripov strežnik. Stražnik nam nato kot odgovor (ang. response) vrne žeton. Knjižnica checkout.js nam zagotavlja, da se občutljivi podatki s kartice ne obravnavajo na našem strežniku. Slika 5.6 prikazuje pridobivanje žetona in pošiljanje zahteve na servis, ki bo poslal zahtevo o plačilu na naš strežnik. Na strežniku se nato zahteva o plačilu pošlje na Stripe (slika 5.7). Ko je zahteva uspešno izvedena, se zgodi preusmeritev na začetno stran.

```
let handler = (<any>window).StripeCheckout.configure({
    key: 'pk_test_4s9nCEVf4AlsjUQieGfE1Ayb',
    locale: 'auto',
    token: function (token: any) {
        // When user enters all the card data create a payment
        const data = {
            token_id: token.id,
            amount: value
        };

        self._markerSvc.chargeCard(data).subscribe(
            data => {
                self._authSvc.updateUserPlan(assets).subscribe(
                    data => self._router.navigateByUrl( '/' ),
                    error => console.error(error)
                );

            },
            err => console.error(err)
        );
    }
});
```

Slika 5.6: Pridobivanje Stripe žetona.

5.4 Vmesnik za izdelavo strani

Glavni del aplikacije je uporabniški vmesnik, ki uporabnikom omogoča izdelavo in urejanje spletne strani. Celoten postopek se začne z dialogom, ki uporabniku omogoča izbiro obstoječe rešitve ali pa izdelavo nove strani. Nadaljuje se s spreminjanjem vsebine in izgleda strani. Konča pa se pri izdelani strani, ki je pripravljena za objavo.

5.4.1 Pregled narejenih rešitev

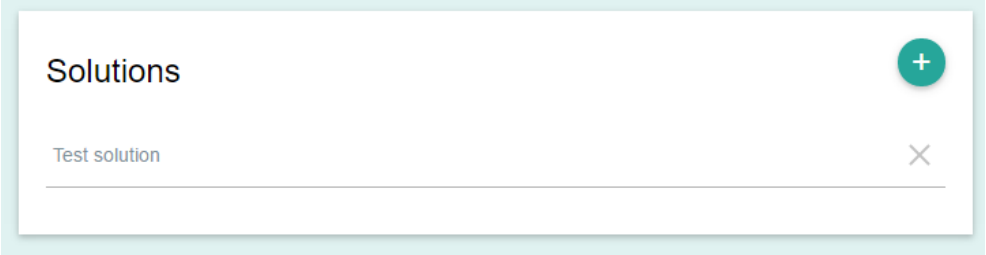
Uporabnik začne s pregledom že narejenih spletnih strani, kjer so izpisane vse rešitve, ki jih je ustvaril. Če želi kreirati novo spletno stran, mora klikniti na gumb + (ustvari - dodaj novo). Ta gumb ga pelje na obrazec, kjer določi ime spletne strani in nato nadaljuje na uporabniški vmesnik za urejanje. Za izbris strani je potreben klik na gumb *X*. Če želi način urejanja, pa je potrebno pritisniti na ime spletne strani, ki jo želi urediti. Slika 5.8 prikazuje pregled

```
let express = require('express');
let router = express.Router();

router.post('/', function(req, res, next) {
    let token = req.body.token_id;
    let amount = req.body.amount;
    //stripe = object generated with our secret key
    let charge = stripe.charges.create({
            amount: amount,
            currency: 'usd',
            source: token,
        },
        function(err, charge){
            if(err){
                return res.status(404).json({
                    title: 'Something went wrong',
                    obj: err
                });
            }
            res.status(200).json({
                title: 'Payment done',
                obj: charge
            });
        }
    );
});

module.exports = router;
```

Slika 5.7: Obravnavanje Stripe zahteve na strani strežnika.

spletnih strani, kjer lahko uporabnik izvaja določene aktivnosti. V primeru da uporabnik ni prijavljen, mu je dostop do pregleda onemogočen, saj brez podatkov o uporabniku ne moremo pridobiti njegovih narejenih rešitev, zato uporabnike, ki niso prijavljeni (goste), preusmerimo na začetno stran.

Slika 5.9 prikazuje poizvedbo po vseh rešitvah določenega uporabnika, glede na njegov *userId*.



Slika 5.8: Pregled ustvarjenih spletnih strani.

```
router.put('/:id', function(req, res, next){...});

//get all solutions by user
router.post('/list', function(req, res, next){
    //Solution = mongo schema
    Solution.find({userId: req.body.userId}, function(err, doc){
        if(err){
            return res.status(404).json( { title: 'Error!', obj: err } );
        }
        if(!doc){
            return res.status(404).json({
                title: 'No solutions',
                obj: {message: 'No solutions found!'}
            });
        }
        res.status(200).json({ solutions: doc });
    });
});
```

Slika 5.9: Poizvedba po vseh uporabnikovih spletnih straneh.

5.5 Urejanje

Glavna funkcionalnost spletne aplikacije je izdelovanje lastnih spletnih strani preko uporabniškega vmesnika. Uporabniški vmesnik omogoča uporabnikom brez znanja spletnih tehnologij razviti lastno spletno stran. To močno zmanjša sredstva, ki bi jih uporabniki porabili za planiranje in izvedbo projektov, saj lahko preprosto ustvarijo spletno stran brez znanja programiranja. Spletne strani, ki jih uporabniki lahko kreirajo, so sestavljene iz štirih delov, ki smo jih poimenovali kontrole:

- **glava** (ang. header),
- **navigacijski meni** (ang. nav),
- **glavni del** (ang. main),
- **noga** (ang. footer).

Na sliki 5.10 je prikazana osnovna struktura spletne strani v obliki HTML, slika 5.11 pa prikazuje izgled strukture v brskalniku. Ko uporabnik naredi novo rešitev, dobi prazno spletno stran, ki jo lahko ureja. Ko gre uporabnik z miško čez posamezen element, se mu pokaže gumb *EDIT* za urejanje vsebine posameznega dela strani. V določenih primerih se pokažeta tudi gumba *ADD* in *DELETE*. To se zgodi samo pri glavnem delu (ang. main), saj ima lahko več delov, ne samo enega.

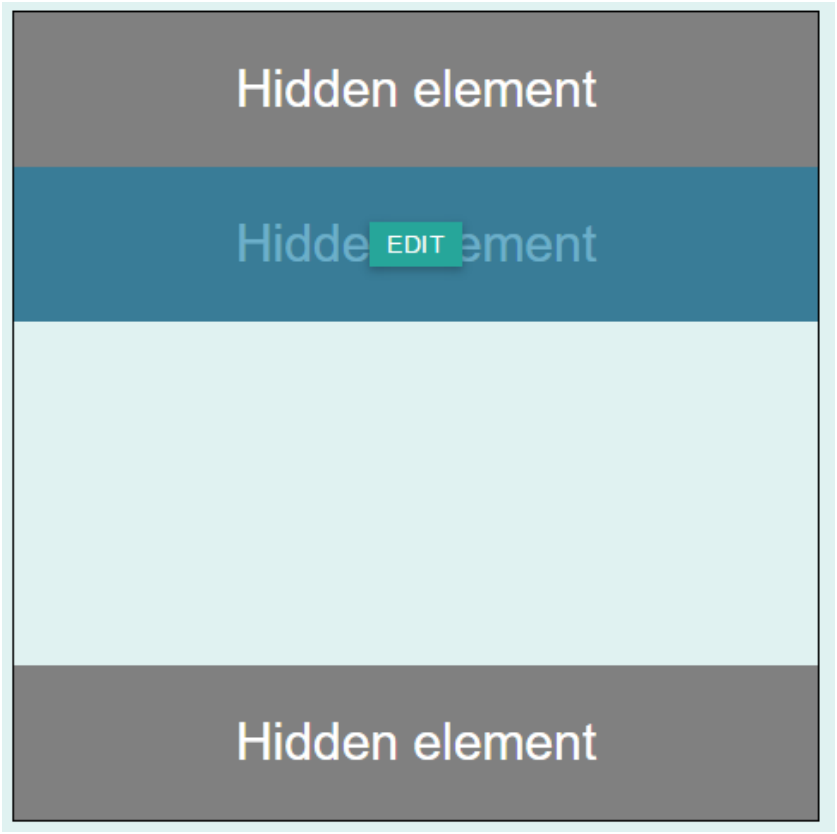
```
<div id="editor-app">
    <dip-header-normal [header]="page.header"></dip-header-normal>
    <dip-nav-normal [nav]="page.nav" [pages]="solution.pages"></dip-nav-normal>
    <dip-main-normal [main]="page.main"></dip-main-normal>
    <dip-footer-normal [footer]="page.footer"></dip-footer-normal>
</div>
```

Slika 5.10: HTML struktura spletne strani.

Ob pritisku na gumb EDIT se uporabniku odpre plošča (slika 5.12), kamor lahko vnese želeno vsebino. Vsaka kontrola ima svojo lastno ploščo za urejanje. Vsaka plošča ima štiri ali pet delov. S klikom na ikone (slika 5.12 zgoraj desno) se prikažejo ali skrivajo sestavni deli plošče.

Sestavni deli plošče za urejanje so:

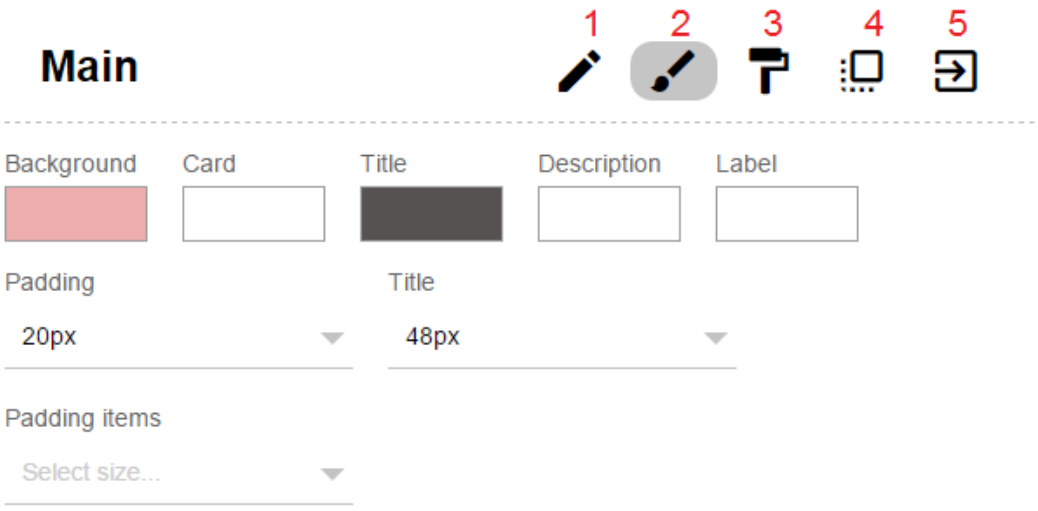
- **urejanje** - pod gumbom številka 1 (slika 5.12) se nahaja urejanje vsebine. Različne kontrole imajo različne možnosti za vsebino. Tu se dodajajo naslovi spletne strani, tekstovna vsebina, itd.
- **izgled** - gumb številka 2 (slika 5.12) je namenjen urejanju izgleda kontrole. Tukaj lahko uporabnik spreminja barve pisave, barve ozadja, odmike na strani (ang. padding), itd.



Slika 5.11: Osnovna struktura spletne strani.

- **teme** - gumb številka 3 (slika 5.12) je viden samo pri *main* kontroli, omogoča pa spreminjanje teme kontrole in s tem pripomore k bolj raznoliki kontroli.
- **postavitev plošče** - gumb številka 4 (slika 5.12) ne vpliva na kontrolo, ampak na ploščo. Privzeta nastavitev je, da se plošča pojavi pod kontrolo, ki jo želimo urejati, s tem gumbom pa omogočimo prosto postavljanje plošče. S tem lahko uporabnik lažje spremlja spremembe, ki jih je naredil. Mi smo ta gumb poimenovali *float/dock mode*.
- **izhod** - gumb številka 5 (slika 5.12) pa enostavno zapre ploščo.

Vse spremembe narejene na plošči za urejanje se v realnem času pokažejo



Slika 5.12: Plošča za urejanje kontrole. Številke v zgornjem desnem delu plošče nad ikonami so dodane za lažjo predstavitev v besedilu.

na kontroli, ki jo urejamo. S takim principom delovanja damo uporabniku občutek hitrega in tekočega delovanja, mu povečamo zadovoljstvo in pokažemo končni rezultat brez nepotrebnega osveževanja strani.

5.5.1 Dodajanje in brisanje strani

Ker je velikokrat smiselno imeti več kot samo eno stran, smo uporabniku omogočili opcijo dodajanja in brisanja strani. Slika 5.13 prikazuje meni, kjer lahko dodamo nove in brišemo obstoječe strani. Do tega menija pridemo s klikom na ime trenutne strani, ki jo urejamo (v tem primeru s klikom na gumb Home). S klikom na gumb + dodamo novo stran k že obstoječim stranem. Novo dodana stran je popoln duplikat prve strani, saj s tem uporabniku privarčujemo nekaj časa, če želi nekatere kontrole ohraniti enake kot na ostalih straneh. Tu mislimo predvsem na kontrole *glava*, *navigacijski meni* in *noga*, saj nekako predvidevamo, da uporabnik želi na vseh straneh enake. V primeru, da si uporabnik želi povsem drugačno vsebino strani, pa lahko to

stori z urejanjem.
Če želi uporabnik izbrisati stran, lahko to stori tukaj s klikom na ikono smetnjaka na zgornjem desnem robu strani, ki jo želi izbrisati. Število strani navzgor ni omejeno, potrebno pa je imeti vsaj eno stran. Da ne pride do napak, smo onemogočili brisanje strani, če ima uporabnik narejeno samo eno stran. Dodali pa smo tudi možnost preimenovanja strani in s tem pomagali k lažji identifikaciji strani.

Slika 5.13: Meni za dodajanje, preimenovanje in brisanje strani.

5.5.2 Stranski meni

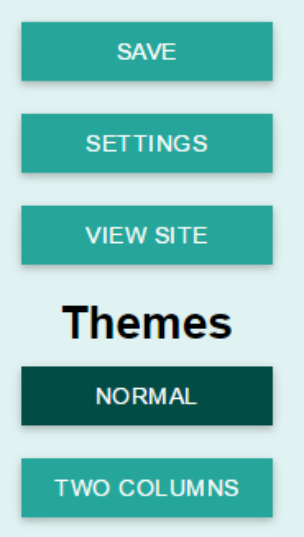
Stranski meni ima pri urejanju spletne strani posebno funkcijo. Tukaj so postavljene opcije, ki vplivajo na celoten izgled strani ali pa so na kakšen drugačen način pomembne za delovanje.

Stranski meni vsebuje naslednje opcije (slika 5.14):

- **Save** - s klikom na ta gumb se shranijo vse spremembe, ki so bile narejene.

- **Settings** - s klikom na ta gumb se nam odpre novo okno (pop-up), kjer lahko nastavimo maksimalno širino strani (stran se centrira na sredino zaslona), ozadje strani (če imamo določeno maksimalno širino, vendar je zaslon širši od določene širine, se doda ozadje), itd.

- **View site** - s klikom na ta gumb v novem zavihku brskalnika odpremo spletno stran, ki smo jo izdelali. To spletno stran lahko vidijo vsi, ne glede na to, ali so prijavljeni ali ne. Te spletne strani se ne da urejati, saj je to končna različica narejene spletne strani.
- **Teme** - teme so namenjene izgledu trenutne strani, ki jo uporabnik ureja. Uporabnik lahko izbira med različnimi temami, ki spreminjajo postavitev komponent.



Slika 5.14: Stranski meni pri urejanju spletne strani.

5.6 Primer izdelane spletne strani

Slika 5.15 prikazuje primer spletne strani po končanem urejanju. Na vrhu strani se nahaja navigacijski meni, preko katerega se lahko dostopa do ostalih narejenih strani ali pa preusmeri na zunanjo povezavo. Navigacijskemu meniju sledi glava s sliko, ter napisom postavljenim na sredino glave. V glavnem delu se nahaja najbolj pomembna vsebina strani. Na sliki 5.15 služi vsebina glavnega dela kot predstavitev storitve, ki jo nudimo. Sestavljena je

iz treh vrstic, ki vsebujejo sliko, naslov in opis storitve. Glavnemu delu pa sledi noga, kjer so vnešene povezave do dodatnih strani in avtorske pravice. Slike uporabljene za predstavitveno stran so bile pridobljene na spletni strani Firebase [11].

Slika 5.15 prikazuje zgolj enega izmed mnogih izgledov, ki ga lahko ustvarimo preko naše aplikacije.

Slika 5.15: Primer izgleda strani po končanem urejanju.

Poglavje 6

Zaključek in nadaljnje delo

V sklopu diplomske naloge je bil narejen prototip sistema, ki uporabnikom omogoča gradnjo lastnih spletnih strani. S tem uporabnikom brez znanja programiranja omogoči izdelovanje spletnih strani brez raznih težav, s katerimi se lahko sooči, kot so izbira tehnologij ali pa izbira ponudnika. Izdelava spletne strani tako ne predstavlja velikega finančnega bremena, prav tako pa uporabnik pri sami izdelavi prihrani čas.

Ob implementaciji smo se spoznali s številnimi novimi tehnologijami, ki se uporabljajo za razvijanje spletnih strani in spletnih aplikacij. Spoznali smo tudi nekaj novih konceptov programiranja, ki nakazujejo nov trend razvoja spletnih strani. Razvoj se je pričel z izbiro funkcionalnih zahtev, ki morajo biti izpolnjene za uporabo sistema. Nadaljevalo se je z izbiro tehnologij, kjer smo se odločili za MEAN, saj omogoča enostavno komunikacijo med odjemalcem, strežnikom in podatkovno bazo. Največja novost je bila kombinacija programskega jezika TypeScript in ogrodja Angular, saj nismo imeli nobenih izkušenj z Angular, ki prinese veliko novih konceptov.

Menim, da smo svoj cilj dosegli, saj je grajenje spletnih strani preko naše aplikacije zelo enostavno in uporabniku ne vzame veliko časa. Rezultati oziroma spletne strani narejene z razvitim vmesnikom pa so, kot kaže tudi predstavljeni primer, dobri. Ostane pa še veliko novih funkcionalnosti, ki bi jih bilo smiselno vgraditi za še večjo uporabnost našega sistema.

6.1 Nadaljnje delo

V prihodnosti bi se spletno aplikacijo lahko izboljšalo z dodatnimi funkcionalnostmi:

- dodatne teme - vpeljali bi večje število tem za ustvarjanje spletnih strani,
- dodajanje lastne domene - uporabniku bi omogočili vnos domene, s tem bi bila ustvarjena spletna stran dostopna na spletnem naslovu (URL), ki si ga je sam določil,
- nadzorna plošča - administratorjem bi se dodalo več funkcionalnosti na nadzorni plošči,
- uvoz strani - uporabnik bi uvozil dokument JSON (narejen po naši predlogi), preko katerega bi aplikacija avtomatsko kreirala spletno stran,
- analitika - beleženje obiskov na spletno stran, pregled narejenih spletnih strani v določenem obdobju, itd,
- lokalizacija - uvedba večjega števila jezikov, s čimer bi povečali potencialni krog uporabnikov.

Literatura

[1] HTML *Wikipedia*. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/HTML. [Dostopano 12. 1. 2017].

[2] Računanje teže CSS selektorjev. [Online]. Dosegljivo: https://css-tricks.com/specifics-on-css-specificity/. [Dostopano 12. 1. 2017].

[3] SASS. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Sass_(stylesheet_language). [Dostopano 12. 1. 2017].

[4] TypeScript dokumentacija. [Online]. Dosegljivo: https://www.typescriptlang.org/docs/handbook/basic-types.html. [Dostopano 12. 1. 2017].

[5] Nabor TypeScript funkcionalnosti. [Online]. Dosegljivo: http://theproactiveprogrammer.com/tag/typescript/. [Dostopano 12. 1. 2017].

[6] Nabor tehnologij MEAN. [Online]. Dosegljivo: http://codecondo.com/7-good-reasons-to-use-mean-stack-in-your-next-web-project/. [Dostopano 12. 1. 2017].

[7] Angular arhitektura. [Online]. Dosegljivo: https://angular.io/docs/ts/latest/guide/architecture.html. [Dostopano 14. 1. 2017].

[8] MongoDB sheme. [Online]. Dosegljivo: https://www.mongodb.com/blog/post/6-rules-of-thumb-for-mongodb-schema-design-part-1. [Dostopano 14. 1. 2017].

[9] Stripe. [Online]. Dosegljivo: https://stripe.com/. [Dostopano 15. 1. 2017].

[10] Stripe Checkout dokumentacija. [Online]. Dosegljivo: https://stripe.com/docs/checkout/tutorial. [Dostopano 15. 1. 2017].

[11] Firebase platforma. [Online]. Dosegljivo: https://firebase.google.com/. [Dostopano 22. 1. 2017].